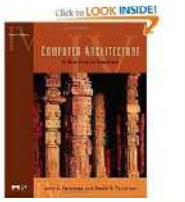
# Sistema de Memoria (1 de 2)

## Índice

1. Introducción a la Jerarquía de Memorias (JM)
2. Memoria cache
3. Memoria principal
4. Memoria virtual

## Bibliografía

- **[MIG]** Mª Isabel García Clemente, *Sistema de Memoria*, Ed. Fundación Gral. de la UPM, Serv. de Publicaciones de la FI-UPM.
- **[P&H]** J. Hennessy & D. Patterson, Computer Architecture: A Quantitative Approach, 4th Edition (Paperback), Ed. Morgan Kaufmann






## Introducción

### Objetivo:

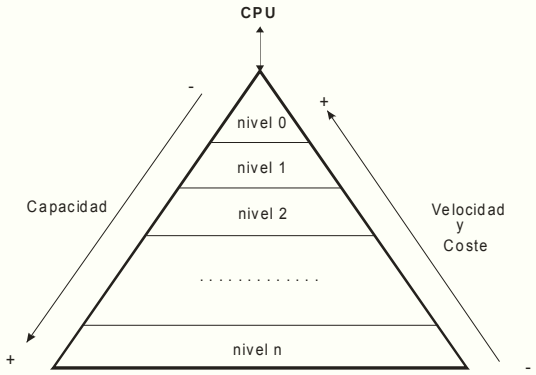
- Almacenamiento de instrucciones y datos ➔ factor determinante en la velocidad de ejecución:

  p.ej., 1 GHz ➔ ¡¡¡1 ns/ciclo!!!
- Necesidad: ↑ *velocidad* + ↑ *capacidad*, pero ↓ *coste*

### Jerarquía de Memorias

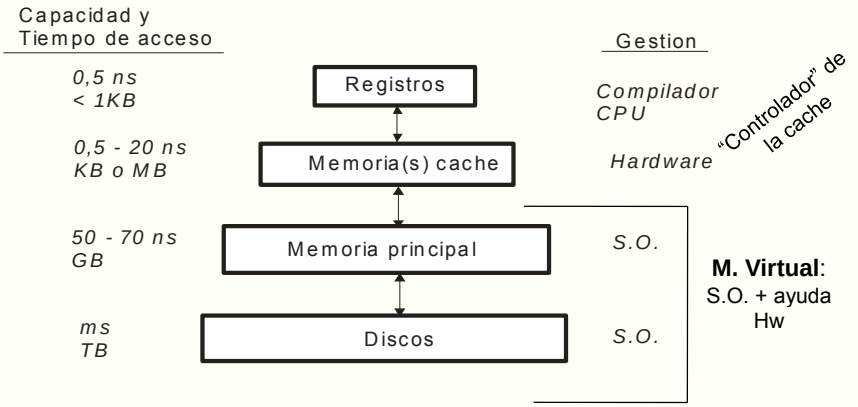
- Múltiples dispositivos de almacenamiento de diferente velocidad, capacidad y coste organizados de forma jerárquica.

## Jerarquía de memorias



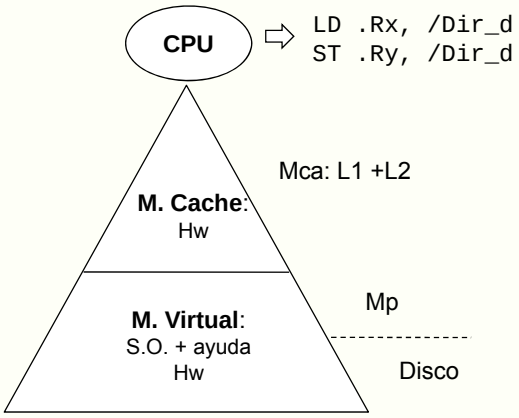

AC 96: Sistema de Memoria 5

## Jerarquía de memorias




AC 96: Sistema de Memoria 6

## JM: ubicación automática de dir. 'Dir_d'






## JM: Principio de Inclusión

- Dj: conjunto de dirs. almacenadas en el nivel 'j' de la JM
- Se cumple entonces el **Principio de Inclusión**:

$$\ldots \subset D_{j-1} \subset D_j \subset D_{j+1} \subset \ldots$$

## JM: Funcionamiento

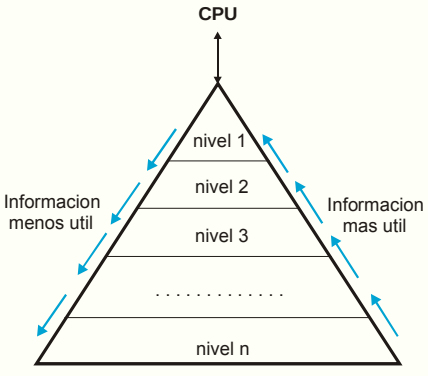
- La CPU solicita acceso a una posición de memoria (`load/store`).
- Se comprueba si la información está en el nivel 1
- Si está
  - Se sirve el acceso desde el nivel 1 (el más rápido)
- Si no está
  - Se comprueba si está la información en el nivel 2
  - Si está
    - Se transfiere la información desde el nivel 2 al nivel 1
    - El nivel 1 transfiere la información a la CPU
  - Si no está
    - Se comprueba si está la información en el nivel 3

. . .

***Nota: se copia siempre un conjunto de dirs. consecutivas: bloques de cache (unos cuantos bytes) o páginas (de orden de KB)***



## JM: Funcionamiento




- Decisiones:
  - Políticas de **extracción**
  - Políticas de **ubicación**
  - Políticas de **reemplazo**
  - Políticas de **escritura**
  - **Traducción** de direcciones

datsi

# JM: Motivación

## Proximidad [*locality*] de Referencias

- Los programas *tienden a* hacer referencia a datos e instrucciones "próximos" a los recientemente utilizados.
  - **Temporal:** Mismas direcciones a las que se ha accedido en un pasado reciente: **[P&H]** '***Regla 90/50 de saltos efectuados***': "90% pa'tras/50% pa'lante"
  - **Espacial:** Direcciones próximas a las que se ha accedido recientemente → **Secuencial**: dirs. consecutivas

*Ejemplo:*

```
sum = 0;
for (i=0; i<10000; i++)
        sum = sum + v[i];
```



datsi

# Proximidad de referencias: traza: ejemplo

- **Traza** de un programa: secuencia (¡que no conjunto!) de dirs. a las accede durante su ejecución.
- Ej. anterior: una posible compilación: 4 bytes/instr., 4 bytes/v[i]
- V[i] desde dir 1000

```
        ORG 0
        LD .R3, #0
        LD .R1, #10000
        LD .R2, #1000
bucle:  ADD .R3, [.R2]
        ADD .R2, #4
        DEC .R1
        BNZ $bucle
```

Traza: 0, 4, 8, 12, 1000, 16, 20, 24, 12, 1004, 16,…

## Proximidad de referencias

Patterson & Hennessy **[P&H]:** *'Regla 90/10 de proximidad':*

"Un programa ejecuta aproximadamente el 90% de sus instrucciones en el 10% de su código"

➔ ¡¡¡La JM funciona gracias al fenómeno de proximidad de referencias!!!

Ejemplos reales: *Visualizing Working Sets*, Evangelos P. Markatos, Institute of Computer Science (ICS), Foundation for Research & Technology - Hellas (FORTH) ;P.O.Box 1385, Heraklio, Crete, GR-711-10 GREECE



## Proximidad de Referencias

- **La proximidad temporal** aconseja:
  - Copiar el contenido de la dirección referenciada al nivel más próximo a la CPU
- **La proximidad secuencial:**
  - Copiar también el contenido de las direcciones cercanas.

**Bloque:**

- Número de palabras consecutivas en memoria.
- Unidad de información que puede estar presente o no en un nivel de la jerarquía.
- Unidad de información que se transfiere entre dos niveles consecutivos de la jerarquía.

## JM: Terminología

- Para un determinado nivel de la 'j' jerarquía:
  - **Acierto:** La información se encuentra en dicho nivel.
    - **Tasa de aciertos** [*hit ratio*] ($Hr_j$): Fracción de los accesos a memoria encontrados en dicho nivel.
    - **Tiempo de acierto** ($T_j$): Tiempo de acceso al nivel.
  - **Fallo:** La información no se encuentra en dicho nivel, hay que buscarla en el siguiente.
    - **Tasa de fallos** [*miss ratio*] = $1-Hr_j$



## JM: Rendimiento

- **Tiempo medio de acceso o Tiempo efectivo:** tiempo medio que se tarda en satisfacer el acceso a una dirección por parte de la CPU: será el valor fundamental para evaluar el rendimiento de una JM (¡no el Hr!)

$$Tef = H_r \times T_j + (1 - Hr) \times T_{fallo}$$

***Ejemplo:***

- 2 niveles: $M_{ca}$ ($T_{Mca}$ = 1ns) y $M_p$ ($T_{Mp}$ = 40ns)
- Bloques de 4 palabras
- Tiempo de fallo = 160 ns
- $Hr_{Mca}$ = 0,95

Tiempo medio de acceso = ¿¿??

## Memoria cache (MC)

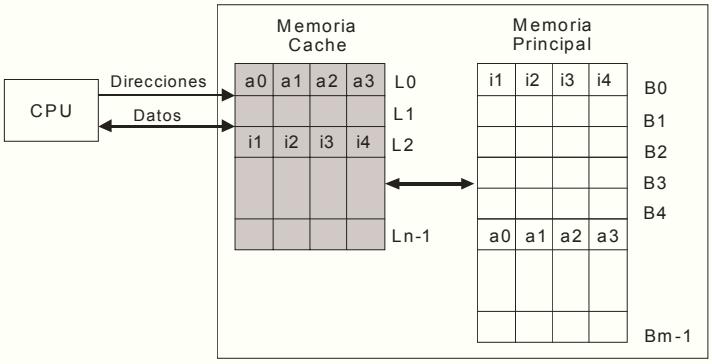
- Memoria rápida de pequeña capacidad, situada entre la CPU y la Memoria principal:
  - $T_{McaL1}$ = Tciclo del procesador, p.ej, 0,5 ns
  - Capacidad de L1: del orden de KB
  - Capacidad de L2: hasta 16 MB
- Contiene parte de la información residente en la Memoria principal:

$$D_{Mca} \subset D_{Mp}$$

→ ***La Mca se gestiona por Hw: es "transparente" al código (todo, sea de usuario o del S.O.)***



## Memoria cache (MC)



## MC: Componentes

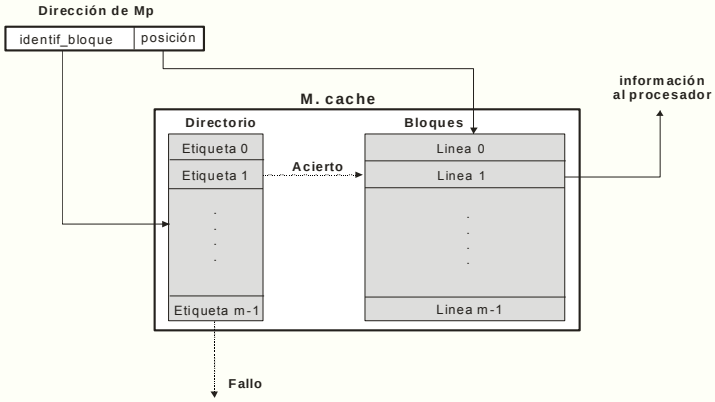
- Mca:
  1. **almacenamiento** ('**líneas**'): donde se albergan físicamente los bloques* se suben del nivel inferior
  2. **directorio**: de alguna forma (dependerá de la pol. de ubicación), qué bloque de Mp está almacenado en cada línea
  3. **controlador** ("el duendecillo") : implementa todas las políticas y realiza el servicio a los fallos: está minuciosamente diseñado para ser muy rápido en sus acciones…

•(*) El almacenamiento se estructura en **líneas**, y cada una contiene un conjunto de dirs. consecutivas de Mp: **bloque de cache**: entre 4 y 64 bytes → el espacio de dirs. de Mp se considera dividido en bloques de igual tamaño



## MC: Búsqueda de bloques

- ¿Cómo se sabe si la información buscada está en la Mca?: la Mca considera las dirs. de la CPU como: nº de bloque + posición dentro del bloque: búsqueda *asociativa* en el directorio: véanse Pols. de ubicación




+ Bit de validez

## MC: Decisiones de diseño

- Política de **ubicación**
  - Dónde se puede almacenar un bloque procedente de Mp
    ⇒ Cómo encontrar la información en Mca
- Política de **extracción**
  - Cuándo y qué bloque se lleva la información a la Mp a Mca
- Política de **reemplazo**
  - Cuando la pol. de ubicación permite elegir, qué bloque se "desaloja" de Mca para albergar al que se sube de Mp
- Política de **escritura**
  - Cuándo se actualiza en Mp una escritura en Mca
- **Tamaño** de la Mca y de sus bloques
  - Cuál es su influencia en la tasa de aciertos y en el tiempo medio de acceso.



## MC: Política de ubicación

⇒ Establece la correspondencia:
bloque Bi de Mp ⇒ línea Lj de Mca

1. **Directa**
   - Cada bloque de Mp sólo puede ubicarse en una única línea de Mca.
2. **Asociativa**
   - Cada bloque de Mp puede ubicarse en cualquier línea de Mca.
3. **Asociativa por conjuntos**.
   - Cada bloque de Mp puede ubicarse en cualquier de las líneas dentro de un único conjunto de Mca: cada conjunto contiene 2, 4, ¿8? líneas

## MC: Política de ubicación

En **[P&H]** se propone que todas las políticas son asociativas, con diferentes grados de "asociatividad":

| | | Asociatividad |
|---|---|---|
| 1. | **Directa:** | 1 |
| 2. | **(Completamente) Asociativa:** | c [nº total de líneas] |
| 3. | **Asociativa por conjuntos:** | S [nº de líneas/set: 2, 4, ¿8?] |

- ¿Cómo afecta el grado asociatividad al $Hr_{Mca}$?

  En general, ↑ asociatividad → ↑ $Hr_{Mca}$

**[P&H]** "*rule-of-thumb*" '**Cache 2:1**': ***"El $Hr_{Mca}$ de una Mca 1-asociativa [Directa] de capacidad C es el mismo que el de una 2-asociativa [2 líneas/set] de capacidad C/2"*** → *¡¡¡Muy útil para la Práctica 2!!!*



## MC: Política de ubicación

Nomenclatura y caso:

Mca:
- **C** = capacidad total de la Mca: 8 KB [$2^{13}$ bytes]
- **Bq** = tamaño de los bloques: 16 bytes [$2^4$ bytes]
- **c** = nº de líneas → c = C/Bq: 512 líneas [$2^9$ líneas]

Mp:
- **M** = capacidad: 4 GB [$2^{32}$ bytes]
- **m** = nº de bloques de Mp → m = M/Bq: $2^{28}$ bloques de Mp

Política de ubicación:

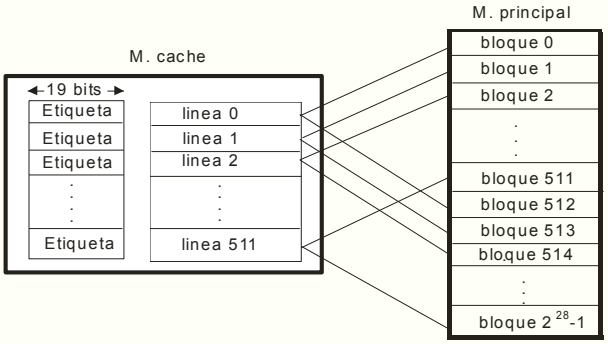
Mp: $Bloque_i$ ⟶ Mca: $Línea_j$

# MC: Ubicación Directa

- Cada bloque de Mp puede ubicarse en una única línea predefinida de Mca:

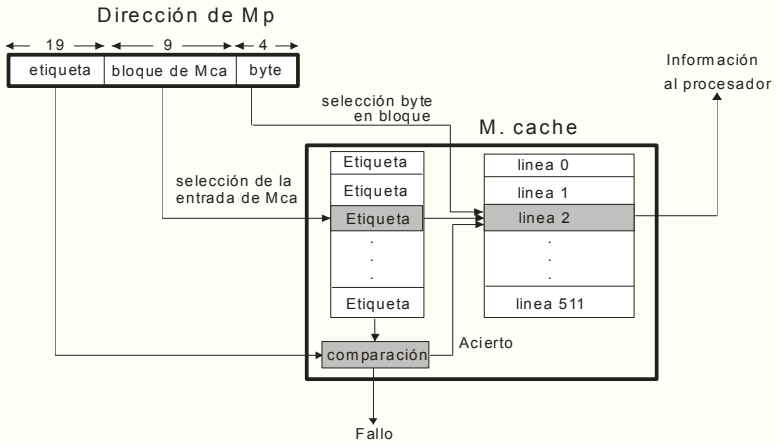
  j := *i mod c* 1 → 1 [1-asociativa]

***Ejemplo:***






# MC: Ubicación Directa

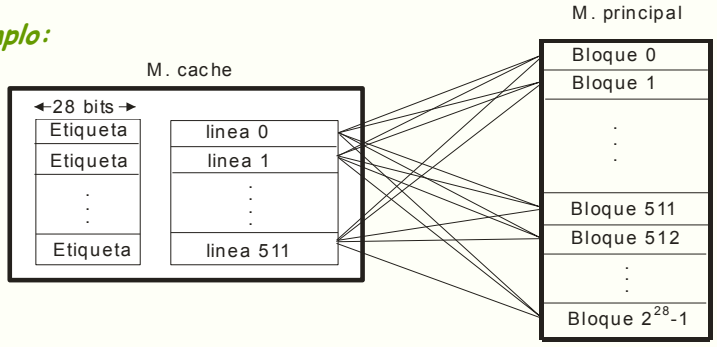



- Ventajas: Sencillez y poco coste y ¡¡¡VELOCIDAD!!!...
- Inconvenientes: Dependiendo de la traza del programa puede producir una tasa de fallos elevada

## MC: Ubicación (completamente) Asociativa

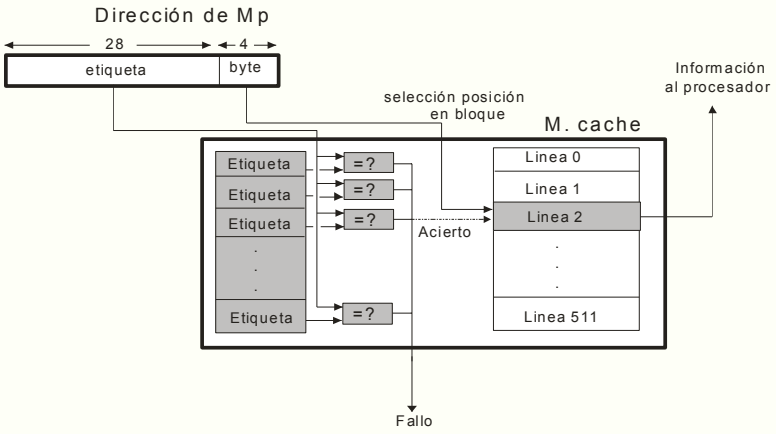
- Cada bloque de Mp puede albergarse en cualquier línea de Mca:
  j := ∀x, donde x: 0..c-1  1 → c [c-asociativa]

*Ejemplo:*






## MC: Ubicación (completamente) Asociativa

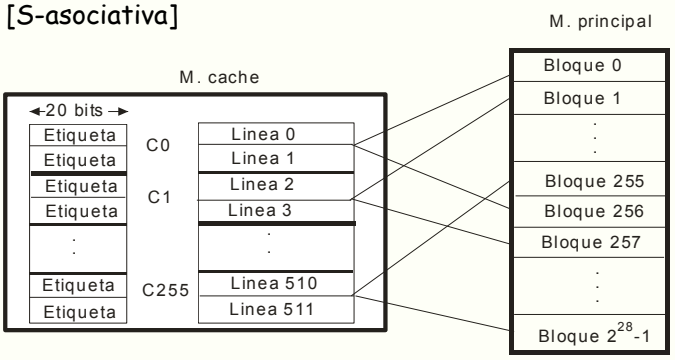



- Ventajas: el mejor $Hr_{Mca}$ con una pol. de reemplazo óptima (¿factible?); la mayor flexibilidad de todas
- Inconvenientes: Complejidad y coste elevado, **¡¡¡LENTITUD!!!**

## MC: Ubicación Asociativa por conjuntos

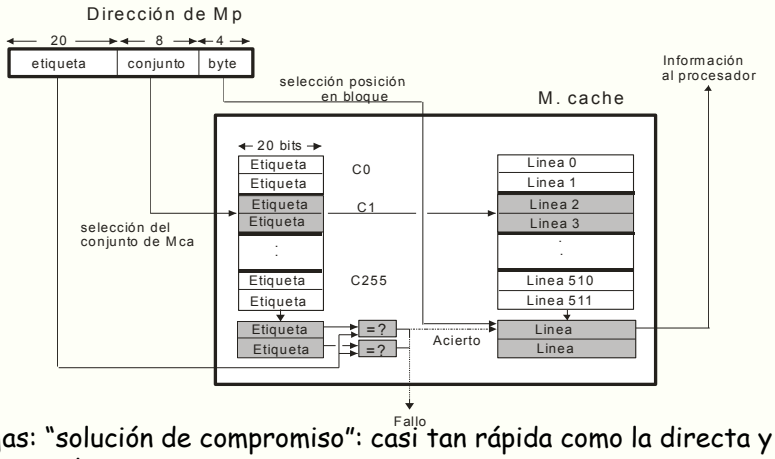
- Cada bloque de Mp puede ubicarse en cualquiera de las línea de un conjunto predefinido de Mca:
  sea **S** = nº de líneas/set → **s** = nº de sets → s = c/S
  j := ∀ de las S líneas en el set *i mod s*
  1 → S [S-asociativa]

***Ejemplo:***






## MC: Ubicación Asociativa por conjuntos




- Ventajas: "solución de compromiso": casi tan rápida como la directa y flexible como la asociativa…

## MC: Política de Extracción

⇒ Decide cuándo y qué bloque se lleva desde Mp a Mca:

1. **Bajo demanda**

   Se lleva a Mca (se "sube") el bloque que produce el fallo: es la pol. "por defecto"

2. **Con anticipación** [*prefetching*]

   Se llevan a Mca bloques que previsiblemente se van a necesitar en un futuro próximo → se fundamenta en el fenómeno de proximidad espacial → ↑ $Hr_{Mca}$



## MC: Política de Extracción

2. **Con anticipación** [*prefetching*] (cont. 1): tipos:

   1. **Siempre** [*always prefetch*]
      Si referencia a $Bloque_i$ ⇒ subir $Bloque_{i+1}$
   2. **Ante un fallo** [*on a miss*]
      Si referencia y fallo en $Bloque_i$ ⇒ subir {$Bloque_i$, $Bloque_{i+1}$}
   3. **Etiquetada** [*tagged*]: especie de "mix" entre 1 y 2
      Si referencia y fallo en $Bloque_i$ ⇒ subir {$Bloque_i$, $Bloque_{i+1}^*$}
      Si referencia a un $Bloque_j^*$ ⇒ {subir $Bloque_{j+1}^*$
      desmarcar $Bloque_j$}

## MC: Política de Extracción

2. **Con anticipación** [*prefetching*] (cont. 2):

**Virtudes**

- Tiende a mejorar el Hr al actuar el pro de la proximidad espacial

**Defectos**

- El *prefetch* puede resultar en vano.
- Presupone una cache 'no-bloqueante' (de lo contrario, no sólo no se gana, sino que incluso se pierde si prefetch en vano).
- Cierta sobrecarga para comprobar que el bloque que se anticipa no está ya en Mca.



## MC: Política de Reemplazo

⇒ Qué bloque se "desaloja" de Mca para albergar a uno que se sube de Mp:

- Implementación (por Hw, como todo en Mca) muy "astuta" para que sea rápida
- Sólo necesarias en caches ***no*** directas: elegir entre 2,4, ¿8?/c
- Tipos: [los mismos estudiados en paginación en S.O.]
  1. **Aleatoria**
  2. **FIFO**
     Se reemplaza el bloque más "antiguo" de los elegibles
  3. **LRU** [least recently used]
     Se reemplaza el bloque al que no se ha accedido desde hace más tiempo

  **2** y **3** requieren almacenar información adicional en Mca y ¡¡¡coste en tiempo de la actualización de "contadores"!!!

## MC: Política de Escritura

⇒ Si hay **acierto** en escritura en Mca, ¿cuándo se actualiza la información en el siguiente nivel?

- **Escritura inmediata** (*write-through*): **WT**
  - Se escribe en Mca y en el siguiente nivel de la jerarquía
    ⇒ Se asegura en todo momento la coherencia entre niveles adyacentes.
- **Escritura aplazada** (*copy-back*): **CB**
  - Se escribe sólo en Mca.
    El bloque de Mca modificado se actualiza en el siguiente nivel sólo cuando es reemplazado.
    ⇒ Bit de modificación/bloque de Mca
    ⇒ Posible problema por no coherencia: Multiprocesadores (MP)



## MC: Política de Escritura

- Cuando hay **fallo** (en escritura) en Mca:
  - **Sin copia en Mca** (*with no allocation*): **WNA**
    - El bloque NO se sube a Mca
  - **Con copia en Mca** (*with allocation*): **WA**
    - El bloque se sube a Mca (como en los fallos de lectura)
- Lo normal es utilizar:
  - Escritura inmediata sin copia: **WTWNA**
  - Escritura aplazada con copia: **CBWA**

## MC: Tamaño de la Mca y de sus bloques

⇒ **Tamaño** de la Mca (C) y de sus bloques (Bq): ambos influyen en la tasa de aciertos y en el tiempo medio de acceso (Tef):

En general, hasta un cierto tamaño(*), ↑Bq ⇒ ↑$Hr_{Mca}$, pero ↑$t_{fallo}$ [los bloques son mayores] ⇒

| ↑$Hr_{Mca}$ vs. Tef |
|---|

(*) Llega un momento en que el Hr empeora (*pollution point*, según [P&G]): Idea: ↑Bq va en contra de la "diversidad" de zonas de memoria: extremo Bq = C …



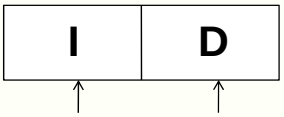
## MC: M Caches separadas I vs. D

Los accesos a direcciones datos (D) y las instrucciones (I) presentan un comportamiento diferente: p.ej., I:

- 75% de accesos (RISC)
- No se modifica
- Más tendencia a proximidad secuencial

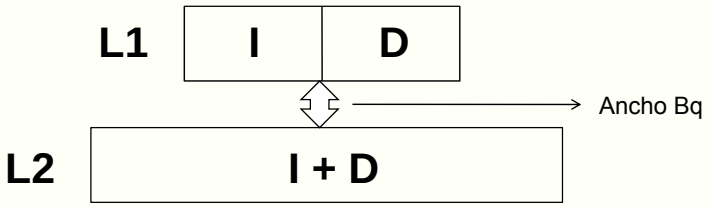
⇒ Caches separadas: ***'Arquitectura Harvard'*** ⇒

- imprescindible para el *pipeline* de instrucciones [véase Tema de Aumento de prestaciones]
- experimentalmente , $Hr_{McaI} > Hr_{McaD}$



## MC: M Cache multinivel

Idea: jerarquía de memorias cache: L1 y L2 (chip CPU): y + L3 en *multicores*




- $Bq_{L1} = Bq_{L2}$
- normalmente se cumple el principio de 'inclusión: DL1 ⊂ DL2



## MC: Memoria cache multinivel

- Consideraciones de diseño diferentes:
  - McaL1: Reducción del tiempo de acierto
  - McaL2: Reducción de la tasa de fallos para reducir el $T_{penalización}$
- Algunos parámetros característicos:

| | Tamaño | Tacierto | Asoc. | Unificada |
|---|---|---|---|---|
| L1 | 16-64KB | 1-3 ciclos | 1-2 | no |
| L2 | 256KB-2MB | 7-15 ciclos | 8-16 | si |
| L3 | 2MB-8MB | 20-30 ciclos | 16-32 | si |

## MC: Medidas de rendimiento

- **Tasa de aciertos** ($Hr_{Mca}$)

  $$\frac{\text{Nº accesos con acierto en Mca}}{\text{Nº total de accesos}}$$

  Objetivo: ≈100%

- **Tiempo medio de acceso** ($T_{ef}$)

  Tiempo transcurrido desde que la CPU realiza una petición al SM hasta que la CPU puede continuar.

  $$T_{acierto} + (1\text{-}Hr) \times T_{penalización}$$

  ≈TMca

- **Tiempo medio de ocupación** ($T_{ocup}$)

  Tiempo transcurrido desde que el SM sirve una petición hasta que puede servir la siguiente.

  $T_{ef} + T_{actualización\ Mca}$

  ≈T entre peticiones de la CPU



## MC: Medidas de rendimiento

**En Caches multinivel**:

- Para **McaL2** (y, en su caso, L3)

  Si fallo McaL1 y acierto en McaL2 ⇒ **Tacc << TMp**

  $$\textbf{Hr Local} = \frac{\text{nº aciertos en McaL2}}{\text{nº accesos a McaL2}}$$

  $$\textbf{Hr Global} = \frac{\text{nº aciertos en McaL2}}{\text{nº accesos de la CPU}}$$

➔ **Hr Global** = % de accesos que van finalmente a Mp: *¡¡¡Más interesante!!!*

## MC: Resumen decisiones vs. rendimiento

| | Tasa de fallos | T. penalización | T. ocupación |
|---|---|---|---|
| Política de ubicación | X | | |
| Capacidad de la Mca | X | | |
| Tamaño de los bloques | X | X | |
| *Victim cache* | | X | |
| *Prefetching* | X | | |
| Caches separads | X | | |
| Caches multinivel | | X | X |
| Buffer de escritura | | X | |
| Políticas de lectura | | X | |
| Caches no bloqueantes | | | X |



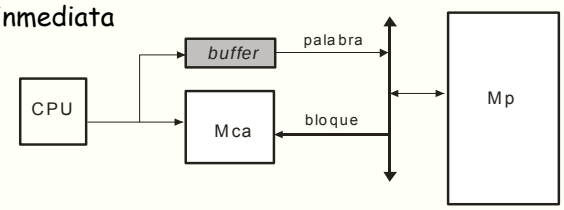
## MC: Algunas mejoras

1. ***Victim cache:***
   Idea: almacena temporalmente bloques desalojados de Mca ➔ si se demanda y se encuentra en la VC no hace falta traerlo de Mp: especialmente útil en fallos por conflicto
2. **Lectura fuera de orden:**
   → caso en que se distingue: $t_{espera}$ vs. $t_{ocupación}$
   fuera de orden : '*out of order*': [¡habitual en problemas del tema!]:
   1º sube primero la dir. demandada a Mca y CPU
   2º luego el resto del bloque

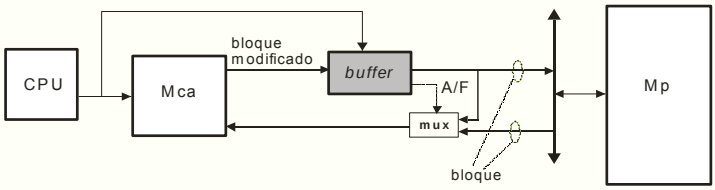
## MC: Algunas mejoras

### 3. Buffer de escritura [*write buffer*]

- Escritura inmediata




- Escritura aplazada






## MC: Algunas mejoras

### 4. Caches no bloqueantes

Idea: deja seguir ejecutando CPU (la "desbloquea") una vez que se ha servido a dir. que ha producido el fallo y mientras sigue la actividad entre la Mca y el nivel inferior debidas al servicio del fallo: p.ej. en:

- prefetching
- lectura fuera de orden

# Sistema de Memoria (y 2 de 2)

## MC: Inicialización y borrado

- La etiquetas de las entradas del directorio siempre tendrán un valor: en cada una se añade un **bit de validez**:

  **V** (bit de validez)/línea

- V se utiliza para invalidar/borrar un bloque → se "resetea" automáticamente con el RESET de la CPU
- Instrucción de la Arquitectura para invalidar la cache entera:

  `Flush_cache`
- ¿Es necesario invalidar la Mca en los cambios de contexto? → ¿y la TLB?

## MC: Tipos de fallos

- Fallos de **primera referencia** (o inevitables o "fríos" o "de calentamiento")
- Fallos por **conflicto**
- Fallos de **capacidad**

Véase la Práctica 2

# Sistema de Memoria (y 2 de 2)

## MC: Inicialización y borrado

- La etiquetas de las entradas del directorio siempre tendrán un valor: en cada una se añade un **bit de validez**:

  **V** (bit de validez)/línea

- V se utiliza para invalidar/borrar un bloque → se "resetea" automáticamente con el RESET de la CPU
- Instrucción de la Arquitectura para invalidar la cache entera:

  `Flush_cache`
- ¿Es necesario invalidar la Mca en los cambios de contexto? → ¿y la TLB?

## MC: Tipos de fallos

- Fallos de **primera referencia** (o inevitables o "fríos" o "de calentamiento")
- Fallos por **conflicto**
- Fallos de **capacidad**

Véase la Práctica 2



## Memoria principal (Mp)

- Mp: "la de siempre": DRAM, SDRAM: DDR2, DDR3,…
- Parámetros:
  1. $t_{acceso}$ **($t_{Mp}$)** o **latencia**: t. que tarda en completar una L/E
  2. **ancho de banda**: nº de bytes/s que se puede transmitir a/desde Mp

Ej. $t_{Mp}$ = 100 ns; **W** [ancho de palabra] = 4 bytes [B]
⇒ 4*10 MB/s
(⇒ "truqui": 100 ns/X ⇒ 10 MX/s)

Objetivo: ↑ancho de banda ⇒ para un W dado, usar '**entrelazado**' → utilidad en la JM: ↓ $t_{bloque}$ entre Mp y Mca:

| | |
|---|---|
| Sin entrelazado: | $t_{subir/bajar\ bloque} \cong Bq \times t_{Mp} + t_{Mca}$ |
| Con entrelazado: | $t_{subir/bajar\ bloque} \cong t_{Mp} + t_{Mca}$ |

# Mp: Entrelazado

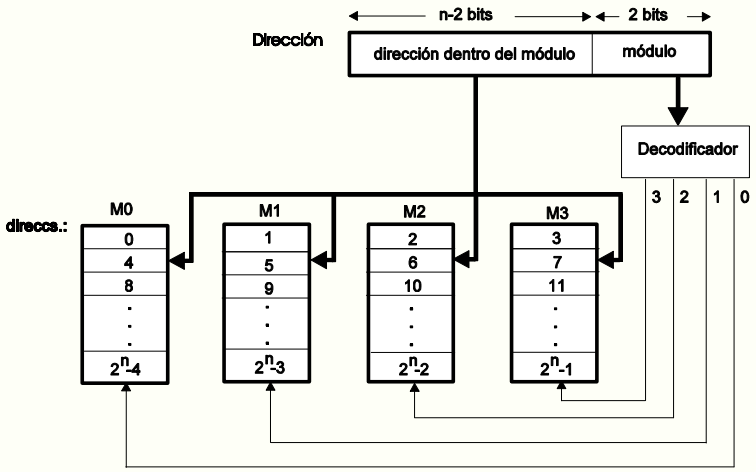
- **M entrelazada** *[interleaved]* : idea: organizar la Mp como K [potencia de 2] **módulos** (de 'orden K') a los que se puede acceder simultáneamente
- Tipos:
  1. Según la distribución de la dirs.:
     1. de orden **inferior**
     2. de orden **superior**
  2. Según cómo se realiza el acceso:
     1. **simple**
     2. **complejo**

Ej.: $K = 4 = 2^2$
$Mp := 2^n$



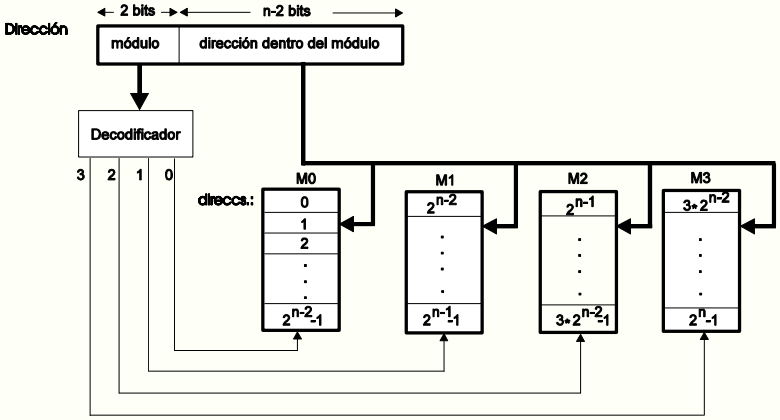
# Mp: Entrelazado de orden inferior

*Idea: módulos consecutivos, direcciones consecutivas*



## Mp: Entrelazado de orden superior

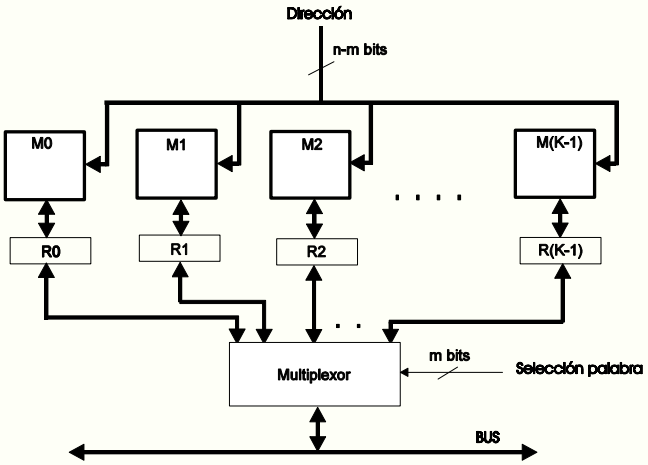
*Idea: dirs. consecutivas en el mismo módulo, con $2^n/K$ dirs. en cada módulo*



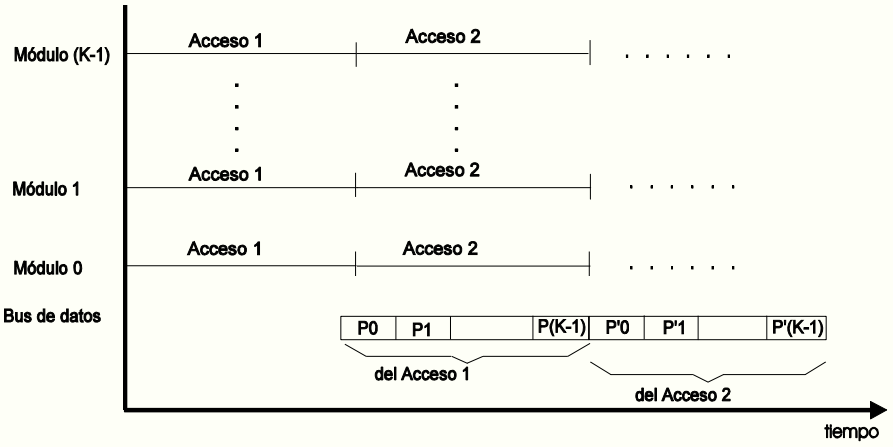



## Mp: Entrelazado simple (+ orden inferior)

*Idea: regs. R0.RK-1 → hacen de k regs. de Datos, D0..DK-1*



## Mp: Entrelazado simple (+ orden inferior)

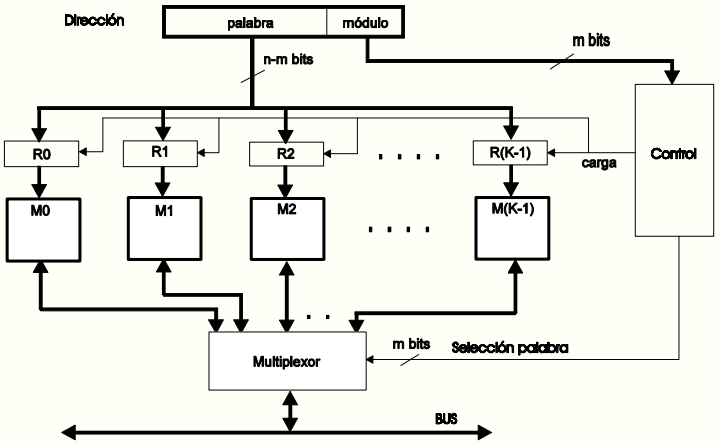



***Entrelazado simple de orden inferior*** → *¡¡¡habitual en los problemas del tema!!!*

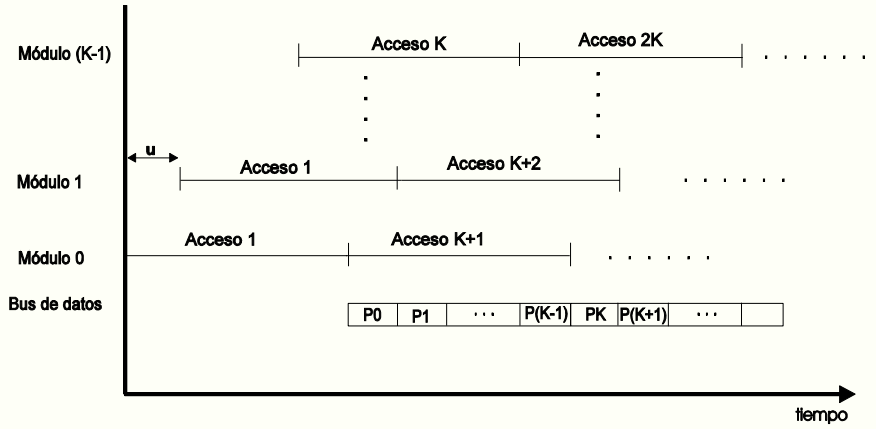


## Mp: Entrelazado complejo (+ orden inferior)

*Idea: regs. R0..Rk-1 → hacen de k regs. de Dirs., A0..Ak-1*



## Mp: Entrelazado complejo (+ orden inferior)




*Mucho más complejos + requieren* ***buses de ciclo partido*** *(que no acabaron de triunfar)*



## Memoria virtual (MV)

- Motivaciones:
  - Mp: recurso escaso
    Capacidad aparente >> Capacidad de la Mp
  - Mp: recurso único: en entornos multiprogramación:
    - Protección y compartición
    - Reubicación
- Soporte físico: JM

| | |
|---|---|
| Mp | • Capacidad aparente "ilimitada" |
| M. Secundaria (Disco) | • En Mp sólo la información útil en cada momento |

datsi

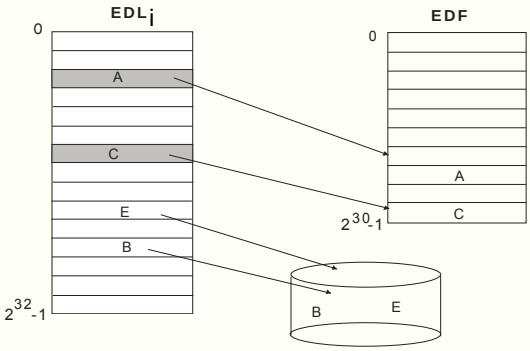
# MV: Espacio de dirs. lógicas vs. físicas

- **EDLi: Espacio de dirs. lógicas o virtuales del Proceso Pi**
  Depende de la Arquitectura (juego de instrucciones)
- **EDF: Espacio de dirs. físicas**
  Depende de la Mp

*Ejemplo:*

Direccionamiento [ri]
Registros de 32 bits
Mp de 1GB
EDL: $0..2^{32}-1$
EDF: $0..2^{30}-1$

Direcciones distintas para la misma información

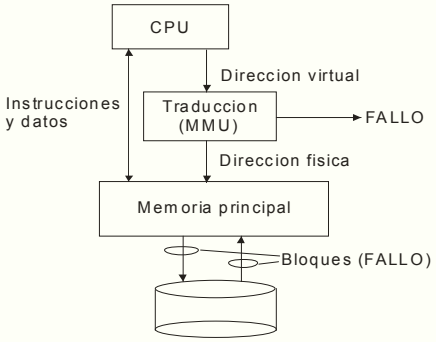





datsi

# MV: Aspectos que se deben tratar

1. **Traducción** de direcciones ⇒ Sw (S.O.) + **Hw**
2. Gestión de los **fallos**
3. **Asignación de Mp** a los procesos



## MV: Implementación

- **Paginación**
  bloques del mismo tamaño: páginas
- **Segmentación**
  bloques de tamaño variable: segmentos
- **Segmentación paginada**
  segmentos que a su vez están compuestos por páginas

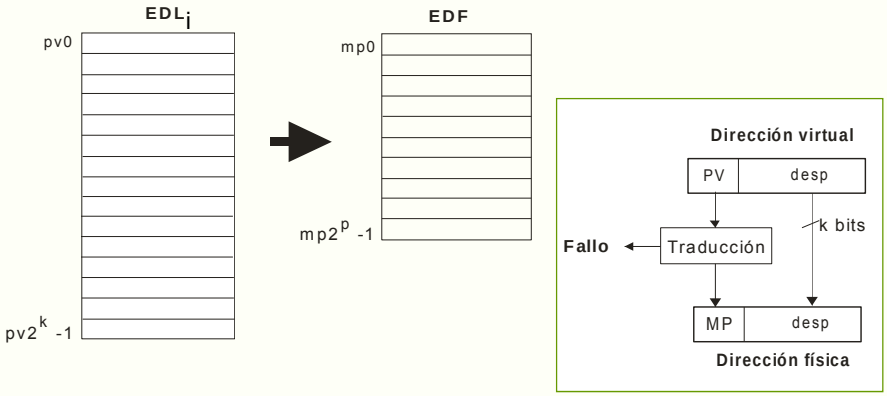


## MV: Gestión de fallos

- Excepción gestionada por el S.O.: **trap de fallo de página** [*page fault trap*: google: nov. '09: ¡¡¡"**3.280.000** de **page fault trap** »!!! + ¡¡¡«**1.320.000** de **page fault exception** »!!! ]
- Acciones a realizar:
  - Suspender el proceso Pi en el que se produjo
  - Ordenar la op. de E/S de transf. del bloque en el que se encuentra la información desde Disco a Mp (DMA).
  - Actualizar la función de traducción.
  - Dar control a otro proceso Pj: cambio de contexto
- Una vez finalizada la transferencia del bloque, y cuando así lo decida el S.O.:
  - "Continuar" la ejecución de Pi:
    - Reiniciar
    - Continuar

# MV: Paginación

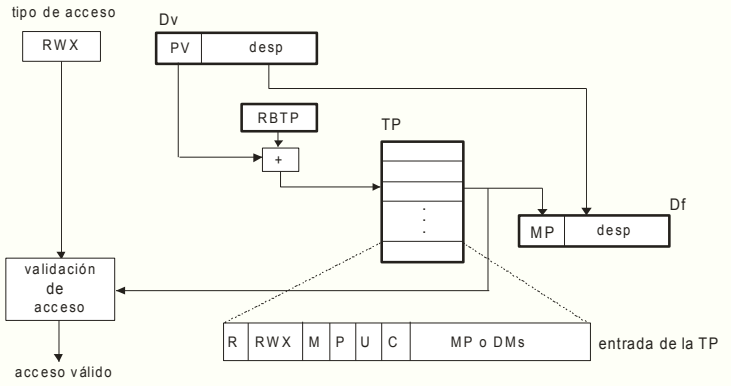
- Correspondencia entre **páginas** [*pages*] virtuales y **marcos de página** [*page frames*]
- Tamaño de páginas $2^k$ bytes






# MV: Paginación

- Implementación de la traducción: **Tabla de páginas**: TPi
- Una tabla por proceso, TPi, + **RBTP**, reg. de la Arquitectura y modo Supervisor



## MV: Tabla de páginas

- TP habitualmente ubicada en Mp.
- Problema: Espacio necesario para su almacenamiento.

*Ejemplo:*

EDL = 4GB

Páginas de 4KB

Nº entradas de la TP = $2^{32}/2^{12} = 2^{20}$

Tamaño de cada entrada de la TP = 4B

⇓

Espacio necesario para la TP de cada proceso = $2^{22}$B = ¡¡¡4 MB!!!

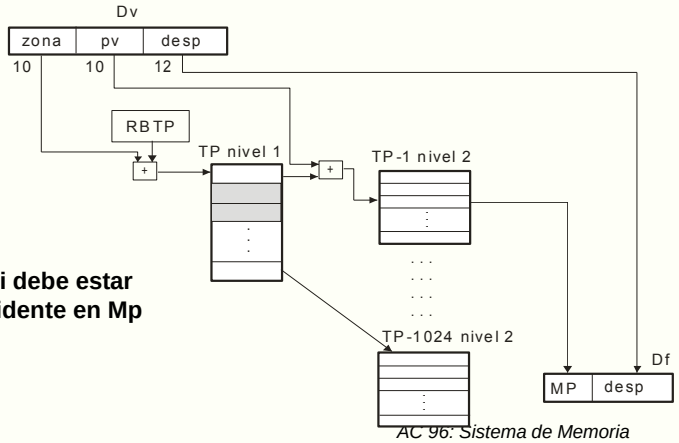
- Pero, afortunadamente:
  - no se utiliza todo EDLi: zonas o franjas, subconjunto pequeño en cada intervalo de tiempo
  - no es un espacio contiguo y varía dinámicamente



## MV: TPs multinivel

- Solución habitual: "paginar la TPi": Tablas multinivel: 3 niveles o más, habitualmente

*Ejemplo:*




**Nota: la TP1i debe estar siempre residente en Mp**

# MV: Aspectos de diseño

- Elección del **tamaño** de las páginas (**P**)
  - Utilización de los marcos de páginas (fragmentación interna)
  - Utilización del dispositivo de memoria secundaria.
    - Tiempo de transferencia:

      Tacceso + (tamaño página/Vtransf)
  - Tamaño de las tablas de páginas: forzar a nº entero de marcos
  - Tamaño habitual: del orden de KB hasta "superpages"
  - Algunos sistemas soportan distintos tamaños
  - Véase http://en.wikipedia.org/wiki/Page_size#Page_size_trade-off
- Política de **ubicación**
- Política de **escritura**
- Política de **extracción**
- Política de **reemplazo**



# MV: Segmentación

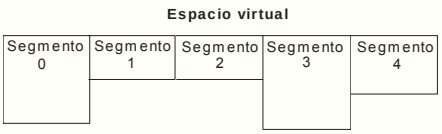
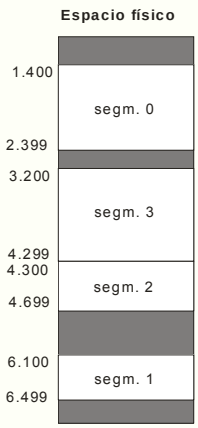



Tabla de segmentos

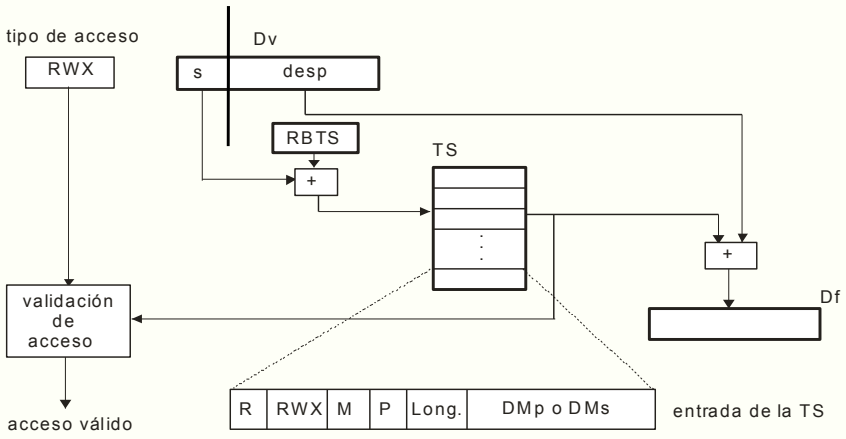
| R | RWX | M | P | Long | DMp |
|---|---|---|---|---|---|
| 1 | R | 0 | 0 | 1000 | 1400 |
| 1 | R | 0 | 0 | 400 | 6100 |
| 1 | R | 0 | 0 | 400 | 4300 |
| 1 | R | 0 | 0 | 1100 | 3200 |
| 0 | R | 0 | 0 | 500 | DMs |




Problema:
Fragmentación externa

## MV: Segmentación

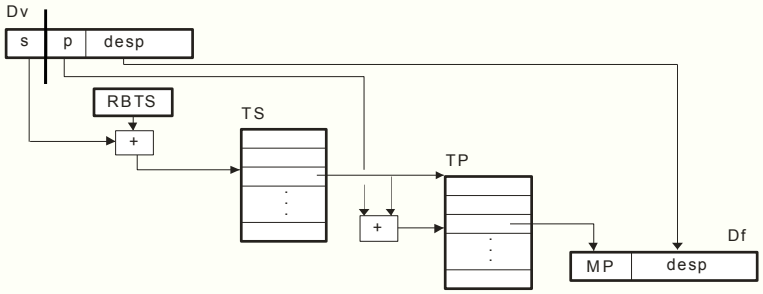
- Traducción de direcciones.






## MV: Segmentación paginada

- Ubicación en Mp de sólo las partes del segmento utilizadas.
- Asignación de bloques de Mp de tamaño fijo (páginas).
- Protección y compartición a nivel de segmento.



## MV: Necesidad de trad. por Hw

- Inconveniente de la Mv: Tiempo empleado en la traducción de direcciones.
- Todas las instrucciones requieren al menos 1 acceso a memoria (*fetch*) y algunas uno o más accesos adicionales.

*Ejemplo:*

ld r1, 0(r2)

2 niveles de TP, Tacc(Mp) = 60 ns

¿Tiempo empleado en accesos a memoria **sin** Mv?

60+60 = **120 ns**

¿Tiempo empleado en accesos a memoria **con** Mv?

2x60+60 + 2x60+60 = **360 ns**



## MV: Aceleración de la traducción

- Mecanismo basado en la propiedad de proximidad de referencias.
- **TLB** (*translation lookaside buffer*)
  - Actúa como una cache de las tablas de traducción:

```
si acierto en TLB
        traducción finalizada
si no
    acceso a las tablas de traducción en Mp
    y actualización de la TLB
```

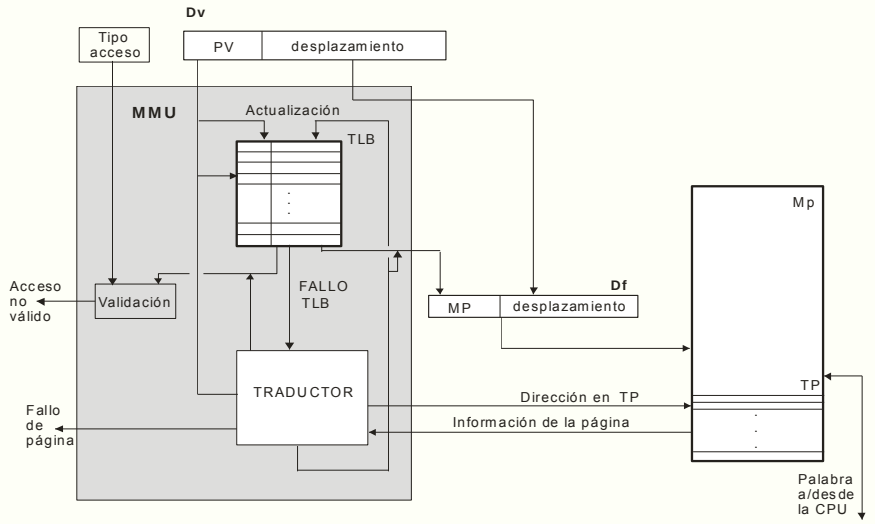
*Ejemplo:*

T(TLB) = 2 ns

¿Tiempo mínimo empleado en accesos a memoria **con** Mv?

2+60 + 2+60 = **124 ns** frente **a 360 ns**

## MV: Aceleración de la traducción: MMU



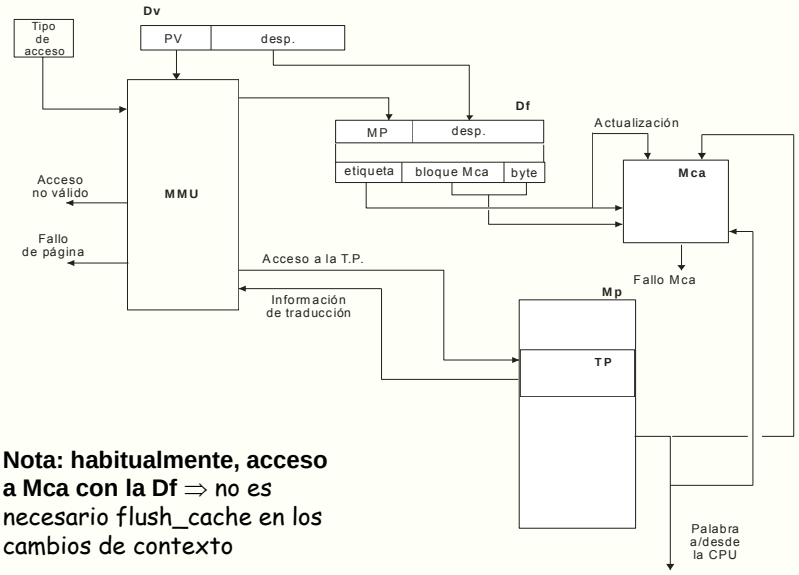



## MV: Aceleración de la traducción: MMU

- Gestión de fallos de TLB:
  1. **Hw**: "máquina de estados": ha sido tradicionalmente así: rápido, pero poco flexible: cierta "programabilidad"…
  2. **Sw**: "la moda": trap de fallo de TLB: rutina con unas pocas instrucciones, un poco más lento que por Hw, pero más flexibilidad: google: nov. '09: "**263.000** de **tlb fault exception » «40.800** de **tlb fault trap**"

- Cuestión: ¿invalidación de la TLB en los cambios de contexto?: añadir el PID: modernamente, **ASID**, *Address Space Identifier*

# JM: Mca + MV




**Nota: habitualmente, acceso a Mca con la Df** ⇒ no es necesario flush_cache en los cambios de contexto